TEXAS TECH UNIVERSITY

David S. Gilliam
Department of Mathematics
Texas Tech University
Lubbock, TX 79409

806 742-2566
gilliam@texas.math.ttu.edu
http://texas.math.ttu.edu/~gilliam

Mathematics 4330/5344 – # 3 Loops, Conditionals, Examples and Programming

1 Introduction

Matlab provides a friendly interactive environment for scientific programming and visualization. Matlab has evolved into a powerful computing environment for developing and testing of models, as well as, obtaining immediate feedback in solving difficult problems. In addition there are numerous toolboxes that greatly expand the potential uses of Matlab. For example, with the C-compiler toolbox you can transform matlab programs into C-code and even executable C-programs. In this lesson I want to continue to introduce you to various basic constructs in Matlab. We will talk about *for loops*, *if statements*, *while statements*, relations such as equal (==) and not equal (~=), less than (<) (or equal <=) and greater than (>) (or equal >=). We will begin to introduce the concept of a matlab m-file (program) and some tools for making these programs interactive, such as, the *input* command. We will talk about the important programming commands *any*, *all* and *find*. Some of the exercises at the end of this lesson are intended to continue to reinforce your ability to build matrices with special structures using builtin Matlab commands such as *diag*, *eye*, *rot90*, *tril*, *triu* and of course *colon notation*.

2 Lesson 3: For Loops, If and While Statements, More Matlab Syntax

In this section we will introduce many useful constructs used over and over again in Matlab programming.

1. The program statement

   ```
   x=input('{some text defining x}')
   ```

   allows you to interactively input the value of `x` during program execution. For example, in the last worksheet you built vectors of given length with given properties. For programming purposes it is useful to allow someone to input any desired length of a vector at the time of program execution.

   ```
   n=input(' input an integer n = ');
   ```

   Then issue the command

   ```
   v=(1:2:n).^2
   ```

2. In matlab programming there are several types of loops and conditionals.

   (a) First we consider the *for loop*. The syntax is

   ```
   for {var} = {a vector of counter values}
   {statements}
   end
   ```

   for example

   ```
   for i=1:3
      x(i)=i^2
   end
   ```

   produces `x=[1,4,9]` .

   (b) Here is an example of nested for loops

   ```
   for i=1:5
     for j=1:5
       a(i,j)=1/(i+j)
     end
   end
   ```

   To do these same operations more efficiently in matlab we can use the following: `aa=hilb (5)`

   (c) Here is an example using a for loop to exhibit the phenomenon of "rounding errors." By computing the values of a sixth degree polynomial in a "dumb" way, then zooming in on a plot of the function, you will see that we get a picture that tells us something very disturbing – namely that a polynomial of degree six appears to have many more than six zeros. The necessity of being careful in programming should be clear from this example. This file is available in the subdirectory *SciCompFiles* which is from [2]. You can change directories to *SciCompFiles* and execute the file by simply typing the name *zoom*. The full name of the file is *zoom.m* – remember all script and function files must end with a "dot m".

```
% Script File Zoom
%
% Plots (x-1)^6 near x=1 with
%increasingly refined scale.
% Evaluation (x-1)^6 via
% x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x +1
% leads to severe cancellation.

   close all
   k=0;
   for delta = [.1 .01 .008 .007 .005  .003 ]
      x = linspace(1-delta,1+delta,100)';
  y = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 ...
       + 15*x.^2  - 6*x + ones(100,1);
      k=k+1;
  subplot(2,3,k)
      plot(x,y,x,zeros(1,100))
  axis([1-delta 1+delta -max(abs(y)) max(abs(y))])
   end
```

3. The next conditional is the *if statement*.

```
 if {relation}   {statements} end
```

   More generally, you can write

```
if {relation}
   {statements}
elseif {relation}
   {statements}
else
   {statements}
end
```

   (a) Here is an example of nested *for loops* and *if statements* First, we use an input statement to select a value of `n`

```
n=4
clear  A
for i=1:n
   for j=1:n
     if i < j
        A(i,j)=-1;
     elseif i > j
        A(i,j)=0;
     else
        A(i,j)=1;
     end
   end
end
A
```

As an illustration of the power of using builtin functions in Matlab programming this matrix can also be built as follows:

```
AA=eye(n)-triu(ones(n),1)
```

4. Next we have the conditional loop `while`

```
while {relation}
    {statements}
end
```

(a) Here is an example

```
 j=1
  while j <= 10
    k(j)=cos(j*pi);
      j=j+1;
  end
k
% compare this with
kk=cos((1:10)*pi)
```

(b) Here is an example from [2]: The problem is to find the smallest positive integer q so that $2^{-q} = 0$ in floating point arithmetic.

```
x=1;q=0;
while x>0
x=x/2;q=q+1;
end
q
```

(c) Here is an example to find the smallest integer p so that $1 + 2^{-p} = 1$ in floating point arithmetic.

```
x=1; p=0; y=1; z=x+y;
while x~=z
y=y/2; p=p+1; z=x+y;
end
p
2*y
eps
```

Note $2 * y$ gives the smallest nonzero floating point number in Matlab *eps*.

5. In the *if* and *while* constructs above we have introduced the notation of a *relation*. A *relation* has the general form

```
{matrix} {relation} {matrix}
```

   (a) The relational operations are

```
== equals
~= not equal
< less than
<= less than or equal
> greater than
>= greater than or equal
```

   (b) The value of a relation is a matrix containing zeros and ones. An entry is `0` if the matrix entry relation is not true and `1` if it is true.

   (c) For example, if

```
A=[1 2;3 4]; B=[1 2;2 4]
T1= A == B
```

   returns the matrix

```
T=[1 1;0 1]
```

   Try these other examples

```
T2= A <= B
T3= A > B
T4= A <= 3
T5= A ~= B
```

   (d) Another useful set of devices used in relational expressions are

```
&  % and,
|  % or
~  % not
```

   These are used on matrices obtained from a relational matrix (as above with all zeros and ones). For example,

```
 T1=[1 1;0 1]; T2=[1 0;0 0]
T=T1&T2
```

gives the matrix

```
 T1=[1 0; 0 0]
```

and

```
 T=T1|T2
```

gives the matrix

```
 T=[1 1;0 1]
```

Finally,

```
 T=~T1
```

gives the matrix

```
 T=[0 0;1 0]
```

6. Two useful programming commands that yield matrices consisting of zeros and ones are *any* and *all*.

   For vectors, `any(V)` returns 1 if any of the elements of the vector are non-zero. Otherwise it will return a 0. For matrices, `any(X)` operates on the columns of X, returning a row vector of 1's and 0's As an example of how you might use the *any* command, let

```
M=floor(11*rand(3,4))-1
any(M>=8)
```

   Here is a program the does the same as the *any* command.

```
v(5:2:10)=5:2:10;
w=0;
k=1;
while (k<=length(v)&w==0)
    if v(k)~=0
       w=1
    end
  k=k+1;
end
```

   This should be compared with

```
any(v)
```

7. Now consider the *all* command. For vectors, `all(V)` returns 1 if all of the elements of the vector V are non-zero. Otherwise it will return a 0. For matrices, `all(X)` operates on the columns of X, returning a row vector of 1's and 0's.

   Here is an example

```
M=floor(11*rand(3,4))-1
all(M<=8)
```

8. The *find* is also very useful in programming. `I = find(X)` returns the indices of the vector X that are non-zero. For example, `I = find(A>100)`, returns the indices of the elements of A that are greater than 100.

   `[I,J] = find(X)` returns the row and column indices of the nonzero entries in the matrix X.

```
x=floor(10*rand(1,20))
I=find(x==3)
J=find(x<5)
```

9. Another useful construct is the *diag* command. If V is a row or column vector with N components, `diag(V,k)` is a square matrix of order `N+abs(k)` (remember `abs` is the absolute value) with the elements of V on the k-th diagonal. $k = 0$ is the main diagonal, $k > 0$ is above the main diagonal and $k < 0$ is below the main diagonal. `diag(V)` simply puts V on the main diagonal. For example,

```
m=3
D= diag(-m:m) + diag(ones(2*m,1),1) ...
 + diag(ones(2*m,1),-1)
```

   produces a tridiagonal matrix of order $2 * m + 1$.

   If A is a matrix, `(A,k)` is a column vector formed from the elements of the k-th diagonal of A. `diag(A)` is the main diagonal of A. `diag(diag(A))` is a diagonal matrix.

```
V=round(10*rand(1,4))
diag(V)
diag(V,1)
```

   and

```
A=round(10*rand(4))
diag(A)
diag(diag(A))
```

10. Finally consider the commands *triu* and *tril* for upper and lower triangular matrices. We will only describe *triu*, see help for *tril*. For a matrix A, `triu(A)` is the upper triangular part of A. `triu(A,k)` is the elements on and above the k-th diagonal of A. $k = 0$ is the main diagonal, $k > 0$ is above the main diagonal and $k < 0$ is below the main diagonal.

```
A=round(10*rand(4))
triu(A)
triu(A,1)
triu(A,-1)
```

11. Finally, we describe an often used command *rot90* which is used to rotate matrix elements. The general syntax is `rot90(A,k)` where k is a positive or negative integer corresponding to a $k * 90$ degree rotation.

```
A=diag(-2:2)
B=rot90(A)
```

ASSIGNMENT 3

1. Write a short Matlab program to input an integer n and build a n by n matrix with the numbers $1, 2, ..., n$ on the main diagonal and zeros everywhere else (hint: Look at the command *diag* .

2. Write a short Matlab program to input an integer n and build the n by n matrix A with entries $a_{ij} = 3^{ij}$. First use loops to do this and then try to redo it without loops.

3. Write a short Matlab program using loops to compute the first 100 Fibonnaci numbers: $a_1 = 1, a_2 = 1, a_n = a_{n-1} + a_{n-2}$

4. Write a Matlab m-file to Input an integer n , a number w and a vector x with n components. Then write a *for loop* program to build $n \times n$ matrices A, B and C with the given entries. Then, try to redo the problem another way using matlab syntax without loops:

   (a) $a_{ij} = w^{(i-1)(j-1)}$

   (b) $b_{ij} = 1/(i + j - 1)$

   (c) $c_{ij} = x_i^{(j-1)}$

   Here is some syntax to input n, w, x.

```
n=input('input an integer n = ');
w=input('input a number  w = ');
x=input(' input an n-vector, x = ')
```

5. In this exercise we consider an example of while loops to solve a problem related to the example in part 4 b) above where we determined the smallest integer q so that $2^{-q} = 0$. Write a short Matlab code to find the smallest positive integer r so that $2^r = \infty$ in floating point arithmetic. Note in Matlab infinity is *inf*. Hint: you might want to use a while statement with: `while x ~= inf`.

6. Write a new program based on the m-file *zoom.m* where you only change the way the function is calculated. Use

   ```
   y=(x-1).^6
   ```

   instead of the expanded version. What is your conclusion as to the best way to evalute this function?

7. The following code generates 100 2×2 matrices with integer coefficients in the ranges $k = 1, 2, \cdots, 20$. For each fixed k it finds the percent that are singular.

   ```
   percent =zeros(1,20);
   for k=1:20
   for i=1:100
      if det(floor((2*k+1)*rand(2) - k)) ==0
          percent(k)=percent(k)+1;
      end
   end
   end
   percent
   ```

   (a) What does the values of the answer *percent* tell you about the percent of matrices that are singular as k increases?

   (b) Repeat the experiment for 3×3 matrices. What can you say about the percent of singular matrices in this case?

   (c) What does this indicate about the percent of singular matrices as the size of the matrix increases?

8. For any n build the $n \times n$ matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & 1 \end{bmatrix}$$

   and find the inverse.

9. Write a program that inputs vectors a and b of the same length n and builds the matrix

$$C = \begin{bmatrix} 1 & a_1 & a_2 & \cdots & a_n \\ 1 & a_1 + b_1 & a_2 & \cdots & a_n \\ 1 & a_1 & a_2 + b_2 & \cdots & a_n \\ \vdots & & & & \\ 1 & a_1 & a_2 & \cdots & a_n + b_n \end{bmatrix}$$

Find the determinant. Take some simple cases. You will see that the answer only depends on the vector b. Can you guess how?

10. Write a program that inputs two numbers a and b and builds the $2k \times 2k$ matrix (you must input k)

$$C = \begin{bmatrix} a & 0 & \cdots & 0 & b \\ 0 & \ddots & \cdots & \iddots & 0 \\ \vdots & & & & \vdots \\ 0 & \iddots & & \ddots & 0 \\ b & 0 & \cdots & 0 & a \end{bmatrix}$$

(i.e., a is on the main diagonal, b is on the backwards diagonal and there are zeros everywhere else.) Find the determinant. Take some simple cases. A formula for the determinant looks like $(a^x - b^y)^z$ for some x, y and z. Find x, y, and z.

11. For any positive integer n build the matrix

$$A = \begin{bmatrix} 1 & 1 & 1^2 & \cdots & 1^{n-1} \\ 1 & 2 & 2^2 & \cdots & 2^{n-1} \\ 1 & 3 & 3^2 & \cdots & 3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & n & n^2 & \cdots & n^{n-1} \end{bmatrix}$$

Compute the determinant. Compare $d = 1!2!3! \cdots (n-1)!$.

Bonus Problems

1. Write a program that inputs a vector a of length n and builds the matrix

$$C = \begin{bmatrix} a_1 & 1 & 0 & \cdots & \cdots & 0 \\ -1 & a_2 & 1 & 0 & \cdots & 0 \\ 0 & -1 & a_3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & -1 & a_n \end{bmatrix}$$

Let C_k denote the determinant of the submatrix of C given by $C(1:k, 1:k)$ for k from 3 to n. Does it appear true that

$$C_k = a_k C_{k-1} + C_{k-2}?$$

2. Write a program that inputs two vectors a and b of length n and then builds the matrix

$$C = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ b_1 & a_1 & a_1 & \cdots & a_1 \\ b_1 & b_2 & a_2 & \cdots & a_2 \\ \vdots & & \vdots & & \vdots \\ b_1 & b_2 & b_3 & \cdots & a_n \end{bmatrix}$$

(Note the size of C is $(n+1) \times (n+1)$.) Find the determinant. Can you guess a formula for the determinant in terms of a and b?

3. Write a program that inputs a vector a of any length n and a number x and builds the $n \times n$ matrix

$$A = \begin{bmatrix} a(1) & x & \cdots & x \\ x & a(2) & \ddots & x \\ \vdots & \ddots & \ddots & \vdots \\ x & \cdots & x & a(n) \end{bmatrix}$$

Then define the function $f(x) = \prod_{j=1}^{n} (a(j) - x)$, compute the derivative of f. The try to compare the determinant of A with $d = f(x) - xf'(x)$ (i.e., write a series of matlab statements that computes f and f' and d – you might want to determine the derivative of f by hand first so you have a formula for it.)

References

[1] *The Matlab Primer,* Kermit Sigmon

[2] *Introduction to scientific computing: a matrix vector approach using Matlab*, Printice Hall, 1997, Charles Van Loan

[3] *Mastering Matlab,* Printice Hall, 1996, Duane Hanselman and Bruce Littlefield

[4] *Advanced Mathematics and Mechanics Applications Using Matlab,* CRC Press, 1994, Howard B. Wilson and Louis H. Turcotte

[5] *Engineering Problem Solving with Matlab*, Printice Hall, 1993, D.M Etter

[6] *Solving Problems in Scientific Computing Using Maple and Matlab,* Walter Gander and Jiri Hrebicek

[7] *Computer Exercises for Linear Algebra*, Printice Hall, 1996, Steven Leon, Eugene Herman, Richard Faulkenberry.

[8] *Contemporary Linear Systems using Matlab*, PWS Publishing Co., 1994, Robert D. Strum, Donald E. Kirk